Contenido

Disparadores

En estos apuntes nos centramos en la funcionalidad de los disparadores en MySql.

> Los disparadores (triggers) son procedimientos almacenados en la base de datos que se ejecutan automáticamente cuando se realiza una operación INSERT, DELETE o UPDATE sobre alguna mesa en concreto.

Es decir, cada tabla al igual que tiene sus propias columnas, sus propios índices etc.
tener, si los definimos, sus propios disparadores (TRIGGERS).
Un TRIGGER está asociado a una tabla, a un evento determinado (INSERT, UPDATE o DELETE) y se ejecuta antes o después de producirse el evento.
Sus aplicaciones son inmensas, como por ejemplo:

- Validar datos
- Mantenimiento de Restricciones de Integridad complejas. Ej: Restricciones de Estado (como el usted sólo puede aumentar).
- Automatizar la generación de valores derivados de columnas.
- Auditoría de una Mesa, registrando los cambios efectuados y cuya identidad los llevó a cabo.
- Lanzar cualquier acción cuando una mesa es modificada.

Características:

- Los disparadores son una forma de extender la funcionalidad al igual que los procedimientos almacenados.
- No requieren intervención del usuario, y se invocan automáticamente.
- Se definen para que se ejecuten antes o después (BEFORE | AFTER) de INSERT, UPDATE o DELETE.

- En MySql actúan por cada fila afectada por los cambios: cada fila insertada o modificada o borrada (ROW). El trigger se ejecuta para cada una de las filas.

• Por cada tabla y evento (INSERT, UPDATE o DELETE), puede haber un máximo de dos disparadores: uno BEFORE y otro AFTER. Por tanto, por cada mesa pueden haber un máximo de 6 disparadores.

Cuando en un determinado evento de una tabla hay definidos triggers el orden de ejecución es el siguiente:

TRIGER BEFORE

EVENTO

TRIGER AFTER

Cualquier error de ejecución en cualquiera de los pasos haría que se parara el proceso y se volviera a el estado anterior anterior al evento.

LIMITACIONES EN EL CÓDIGO DE UN DISPARADOR

- El disparador no puede referirse a la mesa afectada directamente por su nombre. Sin embargo, se pueden utilizar las palabras clave OLD y NEW para referenciar las columnas de la tabla

OLD se refiere a un registro existente que se borrará o que se actualizará antes que esto ocurra.

NEW se refiere a un registro nuevo que se insertará oa un registro modificado después de que ocurra la modificación

- El disparador no puede invocar procedimientos almacenados utilizando la sentencia CALL. (Esto significa, por ejemplo, que no se puede utilizar un procedimiento almacenado para eludir la prohibición de referirse a tablas por su nombre).
- El disparador no puede utilizar sentencias que inicien o finalicen una transacción, tal y como START TRANSACTION, CONMIT, o ROLLBACK.

Manual de referencia MySql 5.0: Capítulo 20. Disparadoras (triggers)

Creación de un disparador

La sintaxis para la creación de un disparador es:

```
CREATE TRIGGER trigger_name
        trigger_time trigger_event
        ON tbl_name FOR EACH ROW
                trigger_body

trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE |
DELETE }
```

Triggers INSERT

Cuando introducimos una fila, lógicamente esa fila no existía antes. Por tanto, nos referiremos a los valores de las columnas que queremos introducir como:

```
NEW. NombreColumna
```

Por cada mesa pueden haber un máximo de dos triggers INSERT: uno BEFORE y otro AFTER.

Una sentencia INSERT puede introducir muchas filas. Para cada una de ellas se ejecutará el código de los posibles TRIGGERS.

En un TRIGGER INSERT nunca podemos referirnos a una columna como OLD.NomColum

Triggers INSERT BEFORE

En un trigger INSERT BEFORE es posible cambiar el valor de las columnas que se van a introducir mediante una instrucción de asignación de valor a una variable. La más común:

```
SET NEW. NombreColumna=ExpresiónVálida;
```

Sólo en los triggers BEFORE se puede cambiar el valor de NEW.NomColumna

Un Trigger INSERT BEFORE nos puede servir, por ejemplo, para:

- Validar datos.
- Mantenimiento de Restricciones de Integridad complejas.
- Automatizar la generación de valores derivados de columnas.

Por ejemplo: Disparador que controle que cuando se introduzca un nuevo empleado en la tabla Empleados de Jardinería el Número, Apellido1 y Apellido2 estén en mayúsculas

```
USE Jardinería;
delimitero //
CREATE TRIGGER Bef_Ins_Emp BEFORE INSERT DONDE EMPLEADOS
FOR EACH ROW
BEGIN
        SET NEW.Número= UPPER(NEW.Número);
        SIETE NEW.Apellido1= UPPER(NEW.Apellido1);
```

```
            SIETE NEW.Apellido2= UPPER(NEW.Apellido2);
END //
/*Probamos su funcionamiento*/


DELIMITER;
INSERT INTO empleados VALUES
 (100,'pepe','Rodríguez','Sánchez','000','pepe@jardineria.net','TAL-ES', NULL, NULL),
 (101,'aNA','Gil','Gil', '000','ana@jardineria.net','TAL-ES', NULL, NULL),
 (102,'iván','Rodríguez','Gil', '000','ivan@jardineria.net','TAL-ES', NULL, NULL);
Select * from empleados where CodigoEmpleado>=100;
```

Triggers INSERT AFTER

Un trigger AFTER normalmente es para realizar algún tipo de auditoría o registro.

Si se ha llegado a este punto es porque no ha habido problemas de ejecución en los pasos anteriores.

En un trigger AFTER NUNCA ES POSIBLE cambiar el valor de las columnas

Por ejemplo: Trigger que introduzca una fila en una tabla llamada RegistroEmpleados cada vez que se introduce una fila en la mesa Empleados donde conste el CodigoEmpleado, la hora en qué se ha producido, el tipo de evento y el usuario que lo ha hecho.

```
USE Jardinería;
/*Primero crear la tabla RegistroEmpleados*/
create table RegistroEmpleados (
NumOperación int primary key auto_increment,
CodigoEmpleado int(11),
hora timestamp default CURRENT_TIMESTAMP,
tipo varchar(10)
usuario varchar(50));
```

```
CREATE TRIGGER Aft_Ins_Emp AFTER INSERT DONDE EMPLEADOS

FOR EACH ROW

INSERT INTO RegistroEmpleados values(null, new.CodigoEmpleado, default,
'INSERCIÓN', user());

/* Fíjate cómo se refiere el CodigoEmpleado que hemos introducido en la
mesa Empleados

No es necesario BEGIN...END ni DELIMITER ya que sólo tenemos una sentencia
*/

/*Probamos su funcionamiento*/

INSERT INTO empleados VALUES

(110,'pepe','Rodríguez','Sánchez','000','pepe@jardineria.net','TAL-ES', NULL, NULL),

(111,'aNA','Gil','Gil', '000','an@jardineria.net','TAL-ES', NULL, NULL),

(112,'iván','Rodríguez','Gil', '000','ivan@jardineria.net','TAL-ES', NULL, NULL);



Select * from RegistroEmpleados where CodigoEmpleado>=110;
```

Ejercicios

1. Trigger que controle que al introducir una fila en Pedidos la DataEsperada y la DataEntrega son posteriores a la DataPedido. Si la DataEsperada es menor que la DataPedido, se cambiará a la DataPedido pasados 3 días. Si la Fecha Entrega es menor que la DataPedido, se cambiará a la DataEsperada.
2. Crea la tabla RegistroPedidos parecida a la del último ejemplo. Trigger que controle que al introducir una fila en Pedidos introduzca una en RegistroPedidos con el CodigoPedido introducido, la fecha y el tipo de operación.
3. Trigger que controle que al introducir una fila en Productos el PrecioVenta sea mayor que cero. Si no lo es, el PrecioVenta será 1.
4. Trigger que controle que al introducir una fila en DetallePedidos la columna Cantidad sea mayor que 0. Si no lo es, se le asigna 1 a Cantidad.
   no puede ser menor que el PrecioVenta del producto. Si es menor, se le asigna el PrecioVenta del producto que está en la tabla Productos.